Arduino and Digital to Analog Converter (DAC)

Author: Nakarin Jayjong

In this tutorial, the demonstration of using a Arduino and Digital to Analog Converter (DAC) as a analog signal generator is presented. The AD5308 DAC and Arduino Nano are used to demonstrated.

Arduino

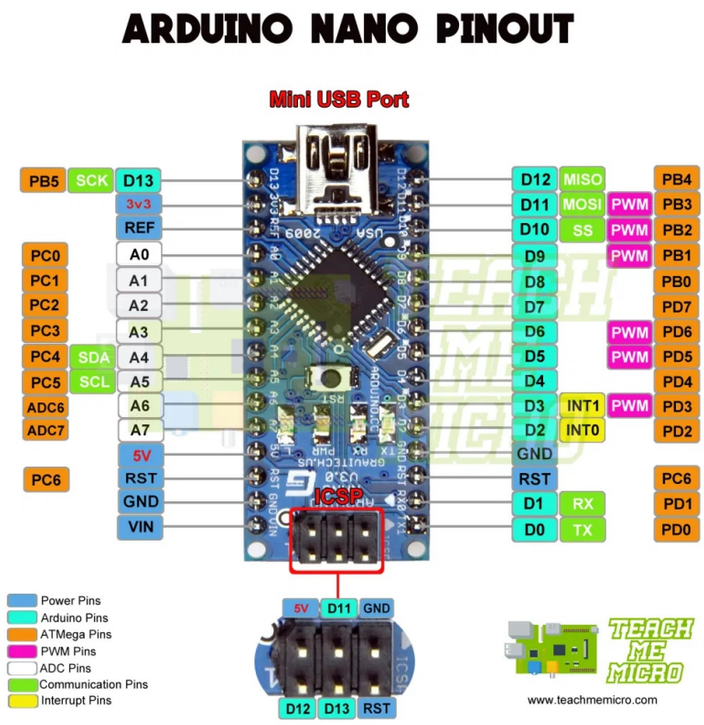

Figure 1. Arduino NANO pinout diagram

The important Arduino pins for this application (SPI communication) are the following:

- D10 This is just a normal digital pin of Arduino. This pin will be used to trigger the $\overline{\text{SYNC}}$ of DAC for the frame synchronization signal when this pin is LOW. Alternatively, this pin can be replaced by other digital pin of Arduino
- D11 (MOSI) This is the "Master Out Slave In" pin.
- D13 (SCK) This pin is serial clock. This pin is connected with a built-in LED. When pin 13 is HIGH, the LED is on and when pin 13 is LOW, it is off. This LED can be used to check if the clock is ticking.

Digital to Analog Converter (DAC)

Digital to Analog Converter (DAC) is a device that allows to translate numeric values into analog signals, so that it can have output voltages variable from 0 to the reference voltage.

The important mnemonics and pins the following:

- $\overline{\text{LDAC}}$ This pin is used to allows any or all DAC registers to be updated when pulse this pin low. Alternatively, this pin can be tied permanently low (tied to GND).
- $\overline{\text{SYNC}}$ This is the frame synchronization signal for the input data. When $\overline{\text{SYNC}}$ goes low, it powers on the SCLK and DIN buffers and enables the input shift register
- SCLK Serial clock input. Data is clocked into the input shift register on the falling edge of the serial clock input.
- DIN Serial Data Input. This device has a 16-bit shift register. Data is clocked into the register on the falling edge of the serial clock input.
- V_{REF} Reference input pin for DACs
- V_{OUT} Buffered analog output voltage

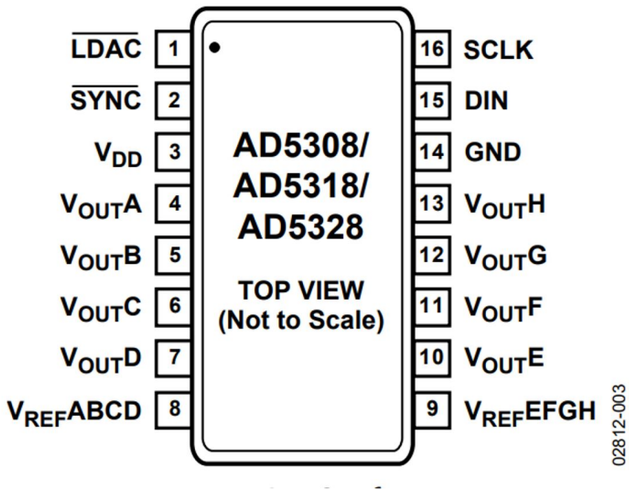

Figure 2. DAC (AD5308) pinout diagram

Serial Interface

In the case of a DAC write, the MSB is a 0. The Address Bits for the AD5308 can be defined by using three binary numbers ranging from 000 to 111 as shown in the Figure 3. The data bits can be represented using eight binary numbers, and LSB should be set to 0 (0000)

A2 (Bit 14)	A1 (Bit 13)	A0 (Bit 12)	DAC Addressed
0	0	0	DAC A
0	0	1	DAC B
0	1	0	DAC C
0	1	1	DAC D
1	0	0	DAC E
1	0	1	DAC F
1	1	0	DAC G
1	1	1	DAC H

Figure 3. Address Bits for the AD5308

Example: In this example, the output of DAC is chosen to be $(175/255) \times V_{ref} = 0.69V_{ref}$ on the pin $V_{OUT}C$, therefore the Address Bits is "010" and the Data Bits is "1010 1111" where MSB is 0 and LSB is "0000". The binary number that will be sent to DAC is "0 010 1010 1111 0000" which can be written in hex number as "0x2AF0" as shown in Figure 4

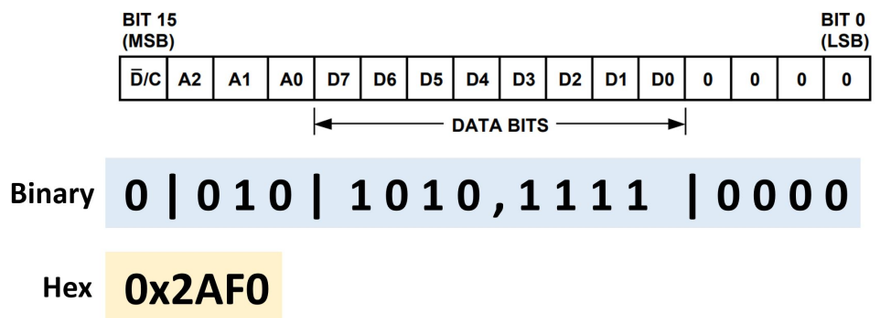

Figure 4. Serial Interface of AD5308

Serial Interface Timing

The serial interface timing for DAC are the following:

- The $\overline{\text{SYNC}}$ pin need to trigger by a pulse to activate the DAC. The minimum high time for $\overline{\text{SYNC}}$ is 50 ns for the AD5308 as shown on CH1 in Figure 5
- The clock will start to tick after falling of the $\overline{\text{SYNC}}$ pulse as shown on CH2 in Figure 5
- After the clock is ticking, the Data Bits will be sent to the DAC as shown on CH3 in Figure 5

Arduino script

This is a Arduino script for varying the analog output of DAC.

```
#include <SPI.h>

void setup() {
    SPI.begin();
    SPI.beginTransaction(SPISettings(16000000, MSBFIRST, SPI_MODE1));
    pinMode(10, OUTPUT);
}
void loop() {
    digitalWrite(10, LOW);
    SPI.transfer16(0x2AF0);
    digitalWrite(10, HIGH);
}
```

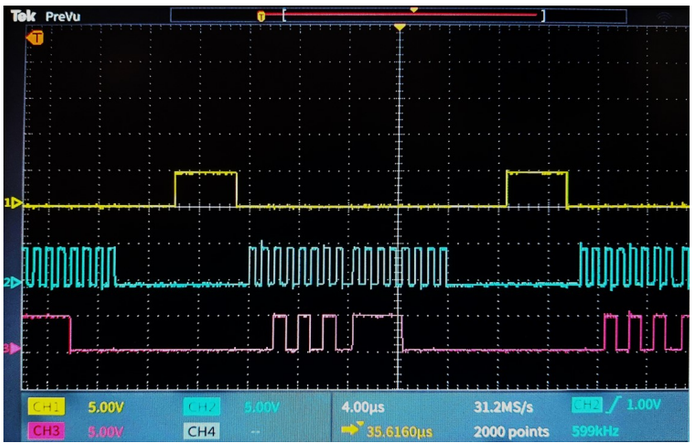

Figure 5. CH1 is $\overline{\text{SYNC}}$, CH2 is SCLK pin, and CH3 is DIN. The Data Bits which is sent to DAC is "0x2AF0" where Address Bits is "010" = "2" and the Data Bits ="1010111" = "AF".

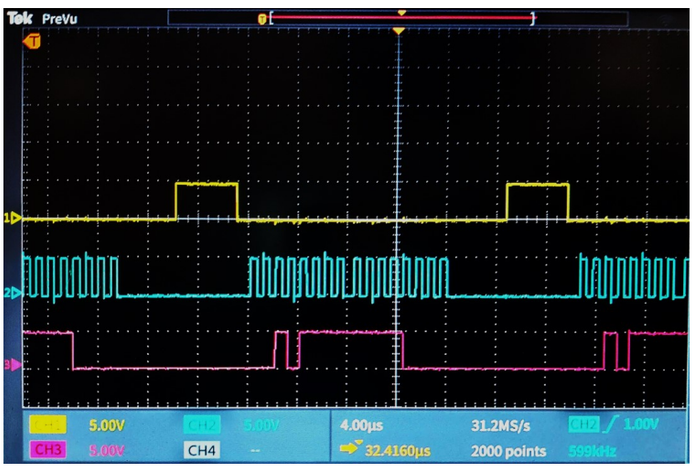

Figure 6. CH1 is $\overline{\text{SYNC}}$, CH2 is SCLK pin, and CH3 is DIN. The Data Bits which is sent to DAC is "0x2AF0" where Address Bits is "010" = "2" and the Data Bits ="11111111" = "FF".

```ino
#include <SPI.h>

void setup() {
    SPI.begin();
    SPI.beginTransaction(SPISettings(16000000, MSBFIRST, SPI_MODE1));
    pinMode(10, OUTPUT);
}

void update_dac(uint16_t ch, uint16_t num) {
// ch is the Channel of the output, num is the DAC number ranging from 0 to 255 (00 to FF)
    digitalWrite(10, LOW);
    SPI.transfer16((ch << 12) | (num << 4));
    digitalWrite(10, HIGH);
}

void loop() {
    update_dac(2, 175);
}
```

Reduction of the $\overline{\text{SYNC}}$ pulse

In last section the $\overline{\text{SYNC}}$ pulse is generated by the "digitalWrite" function of Arduino. The Arduino, however, need some time to comminicate to the chip (ATMega) on the Arduino board. For this reason, changing the PORT register directly can reduce the pulse delay which is generated.

In this example we use pin 10 of Arduino which is equivalent to PB2 of ATMega pin and we can set the pin to LOW and HIGH using the Port Register Control trick as shown in Figure . For more details see [1].

```ino
#include <SPI.h>

void setup() {

    SPI.begin();

    SPI.beginTransaction(SPISettings(16000000, MSBFIRST, SPI_MODE1));
    pinMode(10, OUTPUT);

}

void update_dac(uint16_t Port_ATMega, uint16_t ch, uint16_t num) {
// Port_ATMega is a ATMega pin of the Arduino which is used to SYNC, ch is the Channel of the
     DAC output, num is the DAC number ranging from 0 to 255 (00 to FF)
// In this case pin 10 of Arduino is used and this corresponds to PORTB 2 (PB2) on ATMega pin
    PORTB&= ~(1<<Port_ATMega); // Turn off the PB2
    SPI.transfer16((ch << 12) | (num << 4)); // Transfer Address Bits and Data Bits
    PORTB|= (1<<Port_ATMega); // Turn on the PB2

}

void loop() {
    update_dac(2,2,175);
}
```

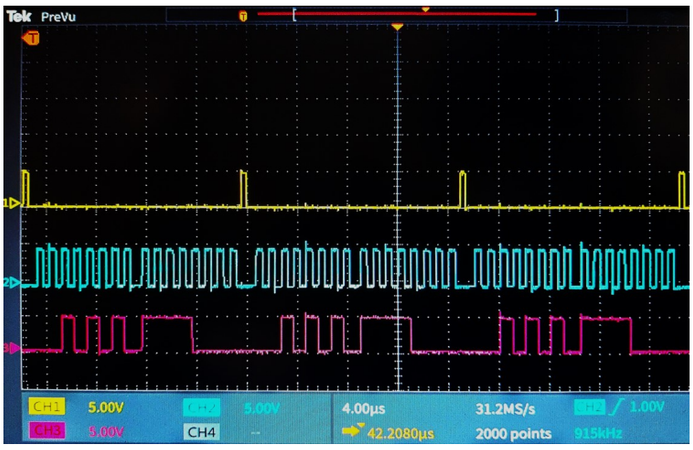

Figure 7. CH1 is $\overline{\text{SYNC}}$, CH2 is SCLK pin, and CH3 is DIN. The Data Bits which is sent to DAC is "0x2AF0" where Address Bits is "010" = "2" and the Data Bits ="1010111" = "AF".

```
PORTB&= ~(1<<Port_ATMega); // Turn off the PB2
SPI.transfer16((ch << 12) | (num << 4)); // Transfer Address Bits and Data Bits
PORTB|= (1<<Port_ATMega); // Turn on the PB2
```

Turn off

PB2 = 0100

Shift bit 1 by 2 (1<<2) ➔ 0001 to 0100. We need to shift by 2 because we use PB2.

~ (1<<2) ➔ ~(0100) = 1011

PORTB&= ~(1<<Port_ATMega) ➔ (0100 & 1011) = (0000) so the pin PORTB is off.

Turn on

PB2 now is = 0000 (turn off)

Shift bit 1 by 2 (1<<2) ➔ 0001 to 0100. We need to shift by 2 because we use PB2.

PORTB|= (1<<Port_ATMega) ➔ (0000 | 0100) = (0100) so the pin PORTB is on.

Note: The underline denote the target bit

Figure 8. Port Register Control for increasing speed of writing of digital pin of Arduino

Ramping the DAC output

The following script is a Arduino script for ramping the DAC output

```
#include <SPI.h>
int t_ramping = 4; // Ramping time in milisecond
int step_t = 80; // The total step in ramping


void setup() {
    SPI.begin();
    SPI.beginTransaction(SPISettings(16000000, MSBFIRST, SPI_MODE1));
    pinMode(10, OUTPUT);
    // Increase the total step to 150 when the t_ramping > 10 milisecond to smooth the output
    // For t_ramping < 10 milisecond the step_t need to be smaller due to the fact that the time delay in each step will never smaller than 2 microseconds
    if (t_ramping>10){
        int step_t =150; // The total step in ramping
    }
}
void update_dac(uint16_t Port_ATMega, uint16_t ch, uint16_t num) {
    // Port_ATMega is a ATMega pin of the Arduino which is used to SYNC, ch is the Channel of the DAC output, num is the DAC number rangibg from 0 to 255 (00 to FF)
    // In this case pin 10 of Arduino is used and this coresponds to PORTB 2 (PB2) on ATMega pin
    PORTB&= ~(1<<Port_ATMega);
    SPI.transfer16((ch << 12) | (num << 4));
    PORTB|= (1<<Port_ATMega);
}
void dac_ramp( uint16_t dac_start, uint16_t dac_end){
    int step_run = 0;
    int dac_run = dac_start; // DAC number which is ranging from dac_start to dac_end
    int tmp;
    // Each DAC update need 2 microseconds
    int time_step = ((float)t_ramping/(int)step_t)*1000-2; // convert into microseconds
    if(dac_start > dac_end)
    tmp = -((int)(dac_start-dac_end)/step_t);
    else
    tmp = (dac_end-dac_start)/step_t;

    while(step_run<step_t){
        step_run+= 1;
        dac_run += tmp;
        update_dac(2, 2,dac_run);
        delayMicroseconds(time_step);
    }
}

void loop() {
    // Ramp up from 0 to 250
    dac_ramp(250, 0);
    // Ramp down from 250 to 0
    dac_ramp( 0, 250);

}
```

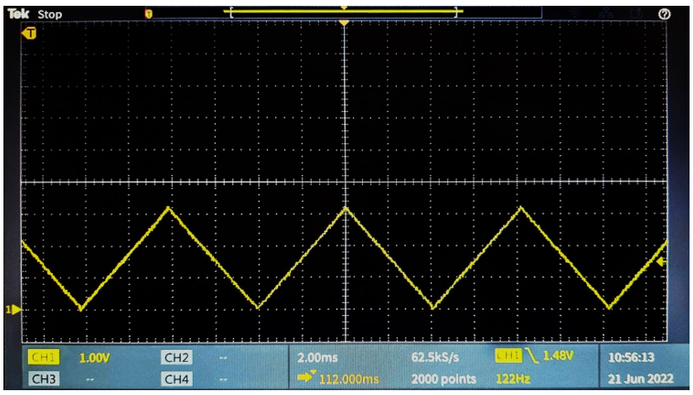

Figure 9. The analog output generated from the ramping code.

References

[1] Electronoobs, "Port register control | increase speed of read/write - arduino101." `https://youtu.be/UhTRrjYXqPU`, 2020.